Closed World Databases Opened Through Null Values

Georg Gottlob* and Roberto Zicari**

*** Institut für Angewandte Informatik, Technische Universität Wien, A-1040 Wien, Austria**
**** Dipartimento di Elettronica, Politecnico di Milano, I-20133 Milano, Italy**

ABSTRACT:

We propose a new approach to the treatment of null-valued attributes in the relational model. The approach is based on the new concept of locally-controlled open world database. A locally-controlled open world database permits the definition of portions of a traditional closed world database as open-world. Attributes, part of a relation, or entire relations can be explicitly defined as "open" through the insertion of null-values. Under this assumption, we consider three different types of null values: the standard *unknown*, *does not exist*, and the new one, called *open*. We give a complete formal specification of the semantics of these null values. We extend the arithmetic and logical operators to cover nulls and outline how relational operators can be extended accordingly. This paper describes ongoing work. We state some open problems to be solved in order to render our approach more operational.

1. Introduction

The treatment of "incomplete" or "inapplicable" information in the relational model has been widely addressed by researchers. One of the major difficulty is given by the fact that several different interpretations can be associated with a null value [ANSI75]. Codd [Codd79] proposed a formal treatment of nulls under the "unknown" (*unk* in the rest of the paper) interpretation. Reiter [Reit86], studied the problem of query evaluation for databases considered as theories of first-order logic with "unknown" marked nulls. Biskup [Bisk81] defined two types of null values which correspond respectively to universally and existentially quantified variables. Lien [Li79], and Zaniolo [Zan83] studied a formal treatment of nulls under the "does not exist" (*dne* in the rest) interpretation. Vassiliou [Vas79] considered the problem of managing nulls with both the "unknown" and "does not exist" interpretations. He pointed out some problems in using a many-valued logic for describing the semantics of arithmetic expressions and logic quantifiers, and proposed an approach based on Scott's denotational semantics. Recently, Codd [Codd86,87] proposed a solution for combining the *dne* and *unk* null values still using a many-valued logic for logical operators.

In the context of an "open world" database [Reit78], Zaniolo [Zan84] introduced the "no information" (*ni* in the rest) interpretation for nulls. A no information null is a lower level value with respect to the *dne* and *unk* interpretations and acts as a placeholder for missing or incomplete information and also characterizes situations where no actual information is available about an attribute. Zaniolo also introduced the concept of a "more informative tuple", that permits to eliminate "less informative tuples" that do not add any information with respect to the tuples already stored in the database. The *ni* interpretation solved some of the problems suffered by Codd's proposal [Codd79]. However some partial knowledge available to the user may be lost with this approach.

Also under the "open world assumption", Roth, Korth and Silberschatz [RKS85] extended Zaniolo's approach considering the *dne* , *unk* , and *ni* all together. They also extended Zaniolo's notion of "more informative" tuple to handle the three types of nulls and defined the semantics of the relational operations accordingly. Other more sophisticated solutions to the problem of null values have been proposed in [Lp79], [ImLp81], [Wo82], [KeW85] and [Im84].

We propose a new approach to the treatment of null values in the relational model. The approach is based on the new concept of locally-controlled open world database. A locally-controlled open world database permits the definition of portions of a traditional closed world database as open-world. Attributes, part of a relation, or entire relations can be explicitly defined as "open" through the insertion of null values. Under this assumption, we consider three types of null values: the standard *unknown* and *does not exist*, and the new one, called *open*.

The most important contribution of our paper consists of presenting a complete specification of the semantics of relational instances which contain null values of either type. Our approach for defining the semantics is a model-theoretic one (see also [Bisk81] and [Mai83]). We will show that to each instance r of a relation schema R, there corresponds a precise set of "possible worlds" which we call the *models* of r. Intuitively, these models stand for all the possible "real world situations" that may correspond to the given relation instance.

The rest of the paper is structured as follows: section two gives a brief general overview of null values under the Open and Closed World Assumptions. Section three informally introduces our new concepts by examples and presents a motivation for our approach. Section four, after stating some basic definitions, describes the semantics of the null values under this new approach. In section five we present a consistency criterium for relational instances and show how some instances can be reduced to smaller ones without loss of information. Section six shows how the semantics of comparison operators and of arithmetical operators can be extended to cover null values and points out some problems related to the generalisation of the relational operators. This paper describes ongoing work. There are still several open problems to be solved before our model can become operational. In section seven, we identify the most important of these open problems and discuss several possible extensions of our work.

2. Null Values under the Closed and Open World Assumptions

Our approach is based upon the observation that when considering a database, the choice between the two well known assumptions, closed and open world [Reit78] is not entirely satisfactory to represent real life situations. The introduction of null values in the relational model stresses this consideration.

We clarify this concept with an example. Consider a simplified company database consisting only of one relation with schema $R(EMP, DEPT, TEL)$ and an instance r_1 of R:

EMP	$DEPT$	TEL
$Smith$	CS	5512

instance r_1

Under the "closed world assumption" (CWA), instance r_1 of relation R says that in our world (our company in the example) one employee, Smith, works in the Computer Science department, and has 5512 as telephone number. No other facts are true: that is, no other employees work for the company, there are no other departments, and Smith does not have a second phone number.

If we consider the same relation R, but under the "open world assumption", things change. With our current knowledge, we can only say that employee Smith works in Computer Science, and has 5512 as telephone number. We do not have any further information whether Smith is the only employee or not, whether CS is the only department, and whether Smith has other telephone numbers.

So far we have considered only totally specified facts (i.e. without incomplete information) stored in the database. When some or all the information regarding attributes of a relation are "missing" or "not known" it is common practice in the database field to introduce null values. A variety of reasons may cause incomplete information in the database, see for example [KeW85] for a description of several sources of incomplete information. Different types of nulls have been considered in the literature [ANSI75].

For closed world databases most authors agree that the various interpretations of nulls can be reduced to the following two:

- *unk*: the value exist, but it is not known;

- *dne*: the value does not exist.

Consider an instance r_2 of relation R:

EMP	$DEPT$	TEL
$Smith$	CS	5512
$Black$	CS	*unk*
$Victor$	CS	*dne*

instance r_2

Under the "closed world" assumption, the following facts can be inferred from r_2:

- there are only three employees: Smith, Black, and Victor;
- there is one department: Computer Science;
- Smith has one telephone: 5512;
- Black has *one* telephone, but we do not know the number;
- Victor does not have any telephone.

No other facts are true.

For open world databases a different type of null, called *no-information*, has been proposed [Zan84] as a lower-level placeholder for both *dne* and unk.

Consider an instance r_3 of R :

EMP	*DEPT*	*TEL*
Smith	*CS*	5512
OBrian	*CS*	*ni*

instance r_3

Under the "open world" assumption the following facts can be inferred from r_3:

- the employees Smith and O'Brian work in Computer Science;
- Smith has 5512 as telephone number;
- we do not know whether O'Brian, in Computer Science, has a telephone or not.

Because of the open world assumption, we also do not know any other information concerning the three attributes of relation R. So for example, we do not know whether Smith has another phone number in CS other than 5512, or whether there exist other employees working for some other departments and so on.

In the next two sections we describe our concept of locally opened databases.

3. Locally Opened Databases

Our approach is simple, yet powerful: We consider closed world databases which can relax part or all of their closure assumption when explicitely marked.

We consider three types of nulls; the standard *unk* and *dne*, and the new one , we call *open*. An *open* null says that an attribute of a particular tuple is under the open world assumption: therefore the attribute value may not exist, or there may be exactly one value, or there may be several values (from the attribute domain) for it; in other words, things are left "open".

The approach is better explained with an example. Consider the three different instances of R, r_a, r_b, and r_c with null values:

r_a :

EMP	*DEPT*	*TEL*
Smith	*CS*	5512
Black	*CS*	*dne*
Victor	*CS*	*unk*

r_b :

EMP	*DEPT*	*TEL*
Smith	*CS*	5512
Smith	*CS*	*open*

r_c :

EMP	*DEPT*	*TEL*
Smith	*CS*	5512
open	*open*	*open*

Instance r_a shows a database where no attributes have the "open" value. This models a closed world database. The semantics of the two null values is the one given in section 2.

Instance r_b models a database which is closed except for the value of attribute *TEL* of the second tuple. Under this interpretation, the following facts can be inferred from r_b:

- Smith is the only employee;
- CS is the only department;
- Smith has one phone number 5512;
- Smith may have zero or one or several other additional phone numbers, we do not know.

No other facts are true.

Note that the value "open" in *TEL* does not mean that *TEL* is a set-valued attribute [AbBi84,ScSc86]. If Smith had more than one additional phone numbers, this would be represented by a set of tuples each with same value for *EMP* and *DEPT*, and different number for *TEL*.

Instance r_c models an open world database. In fact, all attributes of the second tuple are left "open", this corresponds to say that we know that Smith works in CS, and has 5512 as phone number, but nothing else is known and therefore cannot be negated. It is important to note that the way a database is considered (closed, partially-opened, opened) is explicitly controlled through introduction (or absence) of "open" nulls. Hence we are not bound to any partial a priori choice between the open and closed world assumption.

There are three important motivations for our approach.

The first one is the high expressive power resulting from the combination of the three different nulls. The *unk* and *dne* nulls have already been recognized as important by several authors. The addition of the *open* null allows to model a large number of further situations of highly applicative importance.

The second motivation is related to the *nested relation* or NF^2 data model [ScPi82, AbBi84, FiVG85, ScSc86]. We observed that when a single type of null, namely the *unk* null is introduced into a nested relation which is interpreted under the CWA, then this *unk* null must be represented by either *unk* or *open* in the corresponding flat relation (according to the position of the *unk* in the nested relation). Furthermore, when the empty set $\emptyset$ appears as attribute value in a nested relation, then this value corresponds to a *dne* in the corresponding flat relation.

Consider for example the following instance r_n of a nested relation with schema $(EMP, \{TEL\})$. It is easy to see that after application of the UNNEST operator, r_n must be transformed into the following flat instance r_f with the schema (EMP, TEL):

EMP	$\{TEL\}$
a	{123, 124}
b	*unk*
c	$\emptyset$
d	{*unk*}

r_n

EMP	TEL
a	123
a	124
b	*open*
c	*dne*
d	*unk*

r_f

Due to our three different nulls we are thus able to apply (in many cases) the UNNEST operator to a nested relation with *unk* nulls and to model uncertain information in flat relations which otherwise could only be modelled by using the nested relations approach.

The third motivation for our approach is related to the problem of updating views [BaSp81, GPZ88].

Updates of projective views are a source of incomplete information. Consider for example the view V defined as a projection on EMP of the database r:

EMP	*TEL*
Smith	5512
Smith	6514

database r :

EMP
Smith

view V :

Now if we insert into V the tuple < *Zhou* >, this should correspond to adding the tuple < *Zhou*, *open* > to the underlying database r, because we do not know how many phone numbers Zhou has. Note that in most commercial database systems this is not the case: in fact, most systems only have one type of null, the *unk* . Adding the tuple < *Zhou*, *unk* > instead of < *Zhou*, *open* > to the database would introduce an arbitrary and perhaps incorrect mapping between view states and database states: we would constrain Zhou to have only one telephone number.

4. Semantics of Relations with Null Values.

In this section we specify the semantics of relation instances in which the null values *open*, *dne* and *unk* are allowed to appear. Our approach is a model-theoretic one. We will show that to each instance r of a relation schema R, there corresponds a precise set of "possible worlds" which we call the *models* of r. Intuitively, these models stand for all the possible "real world situations" that may correspond to the given relation instance. Our notion of model is comparable to the one of *extension* defined in [Mai83]. However, the *unk* nulls are treated according to the *completion* paradigm as described in [Mai83] and the *dne* nulls represent additional constraints.

This section is subdivided into three subsections. The first introduces our notation and defines some basic concepts. The second subsection gives an exact formal specification of the semantics of relational instances. Finally, our approach is illustrated by several examples in the third subsection.

4.1 Notation and Basic Concepts.

In order to distinguish single values (including null values) from one another we will use in the entire Section 4 the strong equality, also called *symbolic equality*, defined more formally in section 6, denoted by "$==$". If x and y are data or null values then $x == y$ is true iff x and y denote exactly the same value. For instance, the expressions $3 == 3$, $dne == dne$, $open == open$ are all true, while the expressions $3 == 5$, $dne == unk$, $dne == open$, $open == unk$, and $unk == 5$ are all false. The negation of $x == y$ is expressed by $x \not== y$. Later (in Section 6) we will define another concept of equality, called the *semantic equality*. Whenever we use the sign "$=$" in this section, we refer to the usual mathematical equality.

We denote by $R(A_1, \ldots, A_n)$ a relation schema defined over a set of attributes $A_1, \ldots, A_n$. The domain of each attribute A_i is denoted by $Dom(A_i)$. The domain $Dom(R)$ of R consists of the cross product $Dom(A_1) \times Dom(A_2) \cdots \times Dom(A_n)$. We extend each domain $Dom(A_i)$ to an extended domain $Dom^*(A_i)$ by adding the three symbols *dne* , *unk* and *open* which denote the three different null values. The extended domain $Dom^*(R)$ of R consists of the cross product $Dom^*(A_1) \times Dom^*(A_2) \times \cdots \times Dom^*(A_n)$.

A relation instance of a schema R is a subset of $Dom^*(R)$. Such instances are denoted by lower case letters (if necessary with subscripts), such as r, r_1, r_a, and so on.

A *tuple* of an instance r is an element of r. We denote tuples by letters such as t, t', t_1, t_a, and so on. If t is a tuple of an instance r, then $t[A_i]$ denotes the component of t which corresponds to the attribute A_i. The comparison operators $==$ and $=\!\!/\!\!=$ can be extended to apply to entire tuples in the obvious way.

Before giving a precise definition of the concept of *possible world*, let us make some informal remarks.

We wish to establish that to each relation instance r correspond some *models*. Each model is a possible world of the underlying relation schema. In general, there will be several other possible worlds which are not models of r. Each possible world consists of tuples filled with "effective data values" instead of null values. Consider, for example, a relational instance r consisting of one single tuple: $r : \{< a, unk, b >\}$ and assume that the domain of the middle attribute of r is the set of integers $\{1, 2, 3, 4, 5\}$. One possible world which is a model for r is, for example, the set $\{< a, 3, b >\}$. Another model for r is the set $\{< a, 5, b >\}$. Thus models are drawn from relation instances by replacing null values with effective data values. More generally, possible worlds are sets of tuples consisting of effective values.

There is, however, one problem. We do not wish that the null value *dne* be represented by any effective value in a model. This would violate our intuitive understanding of this null value. For this reason, we introduce a dummy symbol $\perp$ which will be used as a "filler" for gaps corresponding to *dne* -nulls (or, in some cases, to *open* nulls). Consider, for example, an instance r consisting of the single tuple $< a, dne, unk >$. Two possible models for this instance are the worlds $w_1 : \{< a, \perp, c >\}$ and $w_2 : \{< a, \perp, d >\}$.

Let us now give a formal definition of the concept of *possible world*. If $R(A_1, \ldots, A_n)$ is a relation schema then a possible world of R is any subset of $Dom^{\perp}(R) = Dom^{\perp}(A_1) \times \cdots \times Dom^{\perp}(A_n)$, where, for $1 \leq i \leq n$ $Dom^{\perp}(A_i) = Dom(A_i) \cup \{\perp\}$. In order to distinguish tuples of possible worlds from tuples of ordinary relation instances, we will refer to the latter by using greek letters such as δ, δ' and so on. The i-th component of such a tuple δ is referred to as $\delta[A_i]$.

Intuitively, a possible world w represents a real life situation where all tuples of w correspond to true facts, while all tuples belonging to $Dom^{\perp}(R) - w$ correspond to false propositions.

The set of all possible worlds of R is denoted by $\Omega(R)$.

Let r be a relation instance. A relation instance $\bar{r}$ obtained from r by textually replacing each occurrence of *unk* by a nonnull data value is called an *unk-completion* of r. This notion is related to the one of *completion* (see [Mai83]).

For instance, let $r = \{< open, 1, unk >, < open, unk, unk >\}$, then two different *unk*-completions of r are: $\bar{r} = \{< open, 1, 2 >, < open, 4, 5 >\}$ and $\bar{r}' = \{< open, 1, 8 >\}$.

The set of all *unk*-completions of a relation instance r is denoted by $UC(r)$.

Given a relation instance r of R and a possible world w of R, we must be able to establish whether w is a model of r or not. Hence, we must provide a precise definition of the concept of model. Once we have such a definition, we can define the *semantics* of a relation instance r (which possibly contains null values) as the set of all models of r. The concept of *model* is thus central to the semantics of relational instances. Its exact definition is given in the next subsection.

4.2 The Model Theoretic Semantics of Relation Instances with Null Values

For all this subsection, let r be a relation instance with schema $R(A_1, \ldots, A_n)$ and let $w \in \Omega(R)$ be a possible world. Our aim is to establish conditions allowing to decide whether w is a model of r or not.

Let t be a tuple of r and let δ be a tuple of w. We say that t *induces* δ , denoted by $t \rhd \delta$, iff for $1 \leq i \leq n$ at least one of the following conditions is satisfied:

$$\begin{cases} t[A_i] == \delta[A_i] & \text{or} \\ t[A_i] == open & \text{or} \\ (t[A_i] == unk \text{ and } \delta[A_i] =\!\!/\!\!= \perp) & \text{or} \\ (t[A_i] == dne \text{ and } \delta[A_i] == \perp). \end{cases}$$

The possible world w is a model of r iff the following four conditions are all satisfied:

1.) $\exists \bar{r} \in UC(r)\ \forall \delta \in w\ \exists t \in \bar{r} :\ t \rhd \delta$.

2.) $\forall t \in r : \ ((t =\!\!/\!\!= < open, open, \ldots, open > \wedge t =\!\!/\!\!= < dne, dne, \ldots, dne >) \Rightarrow \ \exists \delta \in w :\ t \rhd \delta)$.

3.) $\forall \delta \in w\ \neg \exists \delta' \in w : \ (\delta =\!\!/\!\!= \delta' \ \wedge \ \forall 1 \leq i \leq n : (\delta[A_i] =\!\!/\!\!= \perp \Rightarrow \delta[A_i] == \delta'[A_i]))$.

4.) $< dne, dne, \ldots, dne > \in r \ \Rightarrow \ w = \{\}$.

The above four conditions represent the definition of the concept of model which in turn defines the semantics of relational instances with or without null values. Let us give some brief informal comments to each of these conditions.

The first condition states that *unk* nulls are to be replaced each with a single data value – hence they are treated according to the *completion* paradigm [Mai83].

The first condition also states that every tuple of δ must be induced by at least one tuple of r. Intuitively, this means that r has to be interpreted under the closed world assumption unless there appear some *open* nulls in r.

Each tuple with *open* nulls which appears in r may have a wide variety of corresponding induced tuples in w and may thus lead to a local "expansion" of the CWA. In the extreme case, r may contain a tuple t_{open} : $< open, open, \ldots, open >$ which consists only of *open* nulls. In this case, since each possible tuple is induced by t_{open}, the CWA is "expanded" to a maximum extent such that it coincides with the open world assumption. Thus r is *de facto* interpreted according to the open world assumption. Note that *dne* nulls occurring in r may in turn restrict this open world assumption (see condition 3).

The way *open* nulls are treated by our approach corresponds to the *extension* paradigm described by Maier in [Mai83].

The second condition states that each tuple of r (except the two "extreme" tuples) must induce at least one tuple of w. This means that each model of r must contain at least all the "positive knowledge" contained in r and must represent a particular choice for each "uncertainty" expressed by r. Hence r can be viewed as a set of axioms to be satisfied by each model.

In particular, this condition enforces that every tuple of r consisting only of ordinary data values must appear in the model w. Furthermore, every tuple of r containing one or more *unk* null values must induce at least one tuple in w where the *unk* values have been replaced by ordinary data values. The *open* values, in turn, are resolved in each model w, either by replacing them with $\perp$ (a placeholder for non–existent values) or by one or several data values (in the latter case, the tuple of r will correspond to several tuples in w, each with a different choice for the *open* null). Finally, each *dne* null value which appears in a tuple will be replaced by the value $\perp$ in the corresponding tuple(s) of w.

The third condition expresses the strong semantics of the *dne* null value. It says that whenever there exists a tuple δ in w which has some $\perp$ values, then there cannot be any other tuple in w which differs from δ only by the replacement of some $\perp$ values with real data values. Since each occurrence of *dne* in r induces at least one occurrence of a $\perp$ value in w (condition 2), each occurrence of *dne* in r implies the effective nonexistence of real data values in the model w in the given context. For example, if r contains a tuple $< 3, dne, 4 >$, then, by condition 2, w must contain the tuple $< 3, \perp, 4 >$; but then, by condition 3, it is enforced that no tuple $< 3, x, 4 >$ may exist in w, where x is an effective data value.

In a similar way, the third condition also assures that whenever the choice "does not exist" is taken for an *open* value of r, then this is reflected by the nonexistence of other alternatives than $\perp$ for this *open* value in the model w.

Note that it follows from condition 3, that the *dne* value is stronger than the *open* value. This will be made clear by an example which we present in the next subsection.

Finally, the fourth condition says that whenever the "extreme" tuple $< dne, dne, \ldots, dne >$ appears in r, then w must be the empty set. This condition is an additional specification to the semantics of *dne* . It follows from condition 4 and from condition 1 that a tuple of the form $\delta_{\perp} :< \perp, \perp, \ldots, \perp >$ can never appear in any model. Condition 4 assures that the only model which expresses the nonexistence of any data value in a relation is the empty set. In particular, it is enforced that the relation instance $\{< dne, dne, \ldots, dne >\}$ and the empty instance $\{\}$ both have the same unique model $\{\}$. Indeed, we wish that these two instances have exactly the same semantics.

The semantics of a relation instance r can now be expressed as the set $MODELS(r)$ of all models of r. Intuitively, these semantics convey exactly all possible real world situations that may correspond to r. Note that when all domains are finite, then $MODELS(r)$ can be effectively computed.

We are now also able to express the fact that two relation instances r and r' are *semantically equivalent*, denoted by $r \cong r'$:

$$r \cong r' \text{ iff } MODELS(r) = MODELS(r').$$

4.3 Some Examples

Consider a relation schema $R' = (X, Y)$ with $Dom(X) = \{a, b\}$ and $Dom(Y) = \{1, 2\}$. We will present different instances of R' and discuss their semantics.

It is easy to see that each relation instance containing only classical data values admits exactly one model which contains the same tuples as the relation instance itself. This is consistent with our requirement that relation instances without *open* nulls are to be interpreted according to the closed world assumption. The following instance r'_1, for example, admits exactly one model w'_1:

X	Y
a	1
a	2

instance r'_1

a	1
a	2

world w'_1

Let us now consider the instance r'_2 which is obtained by adding the tuple $< open, open >$ to r'_1:

X	Y
a	1
a	2
open	open

instance r'_2

The last tuple of this instance may induce zero or more additional model–tuples, giving raise to a variety of five different models of r'_2 which are:

model $w'_{2a} = w'_1$;

a	1
a	2
b	⊥

w'_{2b}

a	1
a	2
b	1

w'_{2c}

a	1
a	2
b	2

w'_{2d}

a	1
a	2
b	1
b	2

w'_{2e}

Hence $MODELS(r'_2) = \{w'_{2a}, w'_{2b}, w'_{2c}, w'_{2d}, w'_{2e}\}$. This is consistent with our original intention to interpret relational instances which contain a tuple of the form $< open, \ldots, open >$ according to the open world assumption.

In order to illustrate the semantics of *unk* , let us consider the following relation instance r'_3:

X	Y
a	1
a	unk

instance r'_3

Although there are two choices (1 and 2) for the *unk* value, the general rationale for interpreting this instance is still the closed world assumption, since no *open* nulls occur in r'_3. According the conditions 1-4, r'_3 admits the following two models:

a	1

model w'_{3a}

a	1
a	2

model w'_{3b}

Let us now consider a somewhat more complicated relational instance r'_4, which combines several different types of null values.

X	Y
a	dne
b	unk
b	open

instance r'_4

What are the models of r'_4 ? The first tuple of r'_4, by condition 2, enforces the existence of a tuple $< a, \perp >$ in every model of r'_4. By condition 3, no other tuple with X–component a may appear in a model. Moreover, condition 2 enforces that at least one of the two tuples $< b, 1 >$ or $< b, 2 >$ must appear in any model. It follows (by condition 3) that the *open* value of the third tuple cannot be translated into $\perp$. Hence, there remain three alternatives for this *open* value: either 1, or 2 or both. In the latter case, the third tuple would give raise to two tuples in the model; one of these two model–tuples, however, would coincide with the tuple induced by the second tuple in r'_4.

By considering condition 1, we conclude that the set of possible worlds which are candidates for being models of r'_4 is limited to the three worlds:

a	⊥
b	1

w'_{4a}

a	⊥
b	2

w'_{4b}

a	⊥
b	1
b	2

w'_{4c}

Now, by checking all four conditions for each of these worlds, we conclude that all the three are models of r'_4.

Finally, let us define a new relation instance r'_5 of R' by $r'_5 = r'_4 \cup \{< open, open >\}$, i.e., by adding the tuple $< open, open >$ to r'_4.

X	Y
a	*dne*
b	*unk*
b	*open*
open	*open*

instance r'_5

It is easy to see that all models of r'_4 are also models of r'_5, but that r'_5 admits the following two additional models:

a	$\perp$
b	1
$\perp$	2

$w'_{5\alpha}$

a	$\perp$
b	2
$\perp$	1

$w'_{5\beta}$

There are no further models for r'_5, because the choice of the Y–component of the tuple with X–component a is limited to $\perp$. In this sense, we may say that in our model the *dne* null is stronger than the *open* null.

5. Inconsistent Relations and Redundancy Elimination

In this section we will briefly discuss a few particular problems related to the null values introduced by our approach. Some of the issues treated here are still under investigation.

First, let us note that there exist relational instances which do not admit any model. Consider, for example an instance r of R' of the form

X	Y
a	1
a	*dne*

instance r

It is easy to prove that r cannot have any model. Assume r has a model w. Then, by Condition 2 of Section 4, the two tuples $< a, 1 >$ and $< a, \perp >$ must be elements of w, but this is a contradiction to Condition 3.

Indeed, according to our intended informal semantics, r expresses a contradiction: the first tuple states that a is in relation with 1, while the second tuple states that there exists no effective value which is related to "a".

The possibility of contradictory instances due to the *dne* null has also been noted by Roth, Korth and Silberschatz [RKS85].

In general, we will say that a relation instance r is *inconsistent* iff r has no model. Such instances should be avoided, because they have no meaning.

Our definition of inconsistent instance is a semantic one. The question whether there exist simple syntactic criteria for recognizing inconsistent instances arises naturally. In case all domains are infinite, this question is answered positively by the following theorem for which we omit the proof:

Theorem. Let $R(A_1, \ldots, A_n)$ be a relation schema such that $\| Dom(A_i) \| = \infty$ for $1 \leq i \leq n$. A relation instance r of R is inconsistent iff there exist two tuples $t, t' \in r$ such that the following conditions are all satisfied:

a) neither *open* nor *unk* nulls occur in t

b) the *dne* occurs at least once in t

c) t' can be derived from t by the substitution of one or more occurrences of the *dne* null with *unk* or with an effective data value.

This theorem is important, because it guarantees that the recognition of inconsistency can be done by using purely syntactic criteria, i.e., by checking all pairs of tuples of a given instance. It is easy to see, that the complexity of consistency checking is quadratic in the number of tuples of an instance.

The theorem does not hold for finite domains. Algorithms for consistency checking in the case of finite domains are currently under investigation.

Let us now draw our attention to the second issue addressed in this section: *redundancy*.

A tuple t of a relation instance is redundant iff t can be removed from r without changing the semantics of r, i.e., iff $MODELS(r - \{t\}) = MODELS(r)$.

We identify three important situations where a relation contains redundant tuples. These three situations are exemplified respectively by the following instances r_α, r_β, and r_γ :

X	Y
a	*dne*
a	*open*

r_α

X	Y
open	*open*
a	*unk*
a	1

r_β

X	Y
dne	*open*
dne	*unk*
a	b

r_γ

The second tuple of r_α is obviously redundant because there is a unique possibile value for the open null, namely $\perp$. Thus both r_α and $r_\alpha - \{< a, open >\}$ admit exactly one model consisting of the single tuple $< a, \perp >$.

This type of redundancy is called ***type α redundancy*** and is defined more generally as follows: A tuple $t \in r$ is type α redundant iff there is another tuple $t' \in r$ such that t' does not contain *open* or *unk* nulls and t can be obtained from t' by replacing all occurrences of *dne* with *open* .

Let us now consider the second type of redundancy, type β redundancy. It is easy to see that the second tuple of r_β is redundant because each model–tuple induced by the second tuple of of r_β is also induced by the first tuple and because the existence of at least one model–tuple with X–component a is enforced by the third tuple anyway.

Before giving a formal definition of type β redundancy, we introduce a partial ordering "$\sqsubseteq$" on attribute values and on tuples of a relation instance.

Let x and y be nulls or data values. y *refines* x, denoted by $x \sqsubseteq y$ iff one of the following conditions holds:

- x is *open* and y is either *unk* , or *dne* , or a nonnull datavalue;
- x is *unk* and y is a nonnull datavalue;
- x is identical to y.

A tuple $t \in r$ refines a tuple $t' \in r$, denoted by $t' \sqsubseteq t$ iff for each attribute A of r $t'[A] \sqsubseteq t[A]$.

A tuple t of r is type β redundant iff there exist two tuples t' and t'' both distinct from t such that $t' \sqsubseteq t \sqsubseteq t''$ and such that *unk* does not occur in t'.

Finally, let us consider the third pattern of redundancy, type γ redundancy. The second tuple of r_γ is redundant because each model–tuple induced by the second tuple is also induced by the first tuple and vice versa. This is so because the third tuple of r_γ constrains the *open* null appearing in the first tuple to be mapped to a value different from $\perp$ (for consistency reasons).

More generally, we can define type γ redundancy as follows:

A tuple $t \in r$ is type γ redundant iff there exist distinct tuples t' and t'' (different from t), such that t' does not contain *unk* and $t' \sqsubseteq t$ and such that the following three conditions are satisfied for each attribute A of r:

a) $t'[A] == open \Rightarrow (t[A] == unk \wedge t''[A] \notin \{open , dne \}$.

b) $t'[A] == \perp \Rightarrow t''[A] \notin \{unk , open \}$.

c) $t'[A]$ is nonnull $\Rightarrow t''[A] == t'[A]$.

It can be shown that all three types of redundancies are effectively redundancies in the sense of our definition. If finite domains are considered, then there exist other types of redundancies. It remains to be seen whether the here presented types of redundancy α, β and γ are an exhaustive list in case all domains are infinite.

6. Arithmetical, Logical, and Relational Operators

Let us first state some desirable properties of arithmetical and logical operators.

Our four principles for arithmetic and logical operators are the following (we consider in the rest expressions obtained through composition of logical and arithmetic operators):

a) **Extension** : If an expression E contains only nonnull datavalues, then its value is the same as if all operators had their classical meaning.

b) **Preservation of identities:** Any operator (or pair of operators) which is (are) associative, commutative, and distributive for nonnull datavalues have the same property if null values are involved. In particular, we request to hold:

- the associativity and commutativity of $*$, $+$, $\wedge$, $\vee$;
- the distributivity of $*$ over $+$, and of $\wedge$ over $\vee$;
- De Morgan's laws.

However we do not require that 0 is the only neutral element, and that the law of the excluded middle is still valid.

c) **Monotonicity of substitution:** If we replace in an expression E a value x with a value y, such that x $\sqsubseteq$ y, then we obtain a new expression E' whose result *e'* is a *refinement* of the result *e* of E; that is: *e'* $\sqsubseteq$ *e*.

d) **No global information loss for sure values:** If we group all possible datavalues in two groups as follows: group I contains the "for sure" values, i.e., *dne* and all nonnull data values, group II contains "uncertain" values *unk* and *open* , then any expression E which is composed only of values belonging to group I cannot result in a value of group II.

Arithmetic operators

Table 1. displays the semantics of any binary arithmetic operator ($\odot$) in presence of null-valued attributes.

$\odot$	b	dne	unk	open
a	$a\odot b$	dne	unk	open
dne	dne	dne	dne	dne
unk	unk	dne	unk	open
open	open	dne	open	open

Table 1

For particular data values, the choice of the result is left open to the implementor: $a/0$ may result in a run-time error or alternatively in a *dne* value. A different semantics from the one in the above table can also be given to the following expression: $0 * unk$ resulting in 0.

Logical operators

If we place *true* in the top class, *unk* in the second, *false* in the third, *open* in the fourth, and *dne* in the fifth, the logical AND of any two items is an item of whichever class is the lower of the two operands. Table 2 displays the semantics of the AND operator in the presence of null values.

AND	F	T	dne	unk	open
F	F	F	dne	F	open
T	F	T	dne	unk	open
dne	dne	dne	dne	dne	dne
unk	F	unk	dne	unk	open
open	open	open	dne	open	open

Table 2

Note that the result *open* of the expression "F AND *open* " represents an information loss. We know for sure that this expression evaluates to a value different from T for each effective data value the operand *open* may take, but the result *open* stands for the possible values $\perp$, F and T. Nevertheless, *open* is the most appropriate result value we can stipulate for the expression "F AND *open* ". Indeed, each other value would lead to an injustified gain of information. If we used two additional nulls, say $open^-$ which can never be turned into T and $open^+$ which can never be turned into F, we could circumvent this problem. We prefer, however, to cope with a loss of information, rather than dealing with five null values. Note also that a similar problem arises with arithmetical operators. Everybody agrees with us that on real valued domains $unk * unk$ must yield *unk* . But then we loose the information that this result must be a nonnegative number.

If we arrange attribute values in a different order: *dne* as the highest, *open* , *true*, *unk* and *false* next in the order, then the logical OR of any two items is the item which is the higher of the two operands. Table 3 displays the semantics of the OR in the presence of null vlaues. Table 4 displays the semantics of the NOT operator.

OR	F	T	dne	unk	open
F	F	T	dne	unk	open
T	T	T	dne	T	open
dne	dne	dne	dne	dne	dne
unk	unk	T	dne	unk	open
open	open	open	dne	open	open

Table 3

NOT	
F	T
T	F
dne	dne
unk	unk
open	open

Table 4

Note that for AND and OR, *dne* is an "absorbing" truth value, this means that when *dne* occurs in a logical expression, the entire expression results in *dne* . Here *dne* has the semantics of the null-value defined by Bochvar in the context of many-valued logics [Res69, Got82].

It is easy to see that all four principles stated at the the beginning of this section are satisfied by our definition of the arithmetical and logical operators.

Equality and Relational Operators

We distinguish between two kinds of equality of attribute values: a *semantic* equality, denoted with " = ", and a *symbolic* equality, denoted with " == ". The " =" operator checks for semantic equality, while " == " checks for equality of representation, i.e. the operator evaluates to true iff two values are symbolically equal. Table 5 and 6 display the semantics of these operators. The need for two equality representations has also been recognized in [Codd86, GZC87].

=	b	dne	unk	open
a	a=b	F	unk	open
dne	F	T	F	open
unk	unk	F	unk	open
open	open	open	open	open

Table 5

==	b	dne	unk	open
a	$a = b$	F	F	F
dne	F	T	F	F
unk	F	F	T	F
open	F	F	F	T

Table 6

We now discuss the behavior of relational operators when null values are considered. We do not provide formal definitions of the semantics of all relational operators, but limit ourselves to an informal presentation of some of the implications that the introduction of nulls have on the extension of standard operators. A deeper analysis of this part is important and is subject to current research.

Selection σ. Let us restrict our attention to select predicates of the form "$A = d$" or "$A = B$", where A and B are attribute names and d id a nonnull value. We first define a strong selection operator $\hat{\sigma}$ based on the symbolic equality sign which accepts only sure tuples:

$\hat{\sigma}_{A=d}(r) = \{t \in r \mid t[A] == d\}.$

$\hat{\sigma}_{A=B}(r) = \{t \in r \mid t[A] \text{ is nonnull } \wedge t[A] == t[B]\}.$

If we want to define a weak selection $\tilde{\sigma}$ accepting also uncertain tuples, then we are faced with an interesting problem: sometimes an *unk* or *open* null cannot be replaced by a particular value, because this would lead to an inconsistency. For example, if r contains two tuples $< a, unk\ , dne\ >$ and $< a, 4, 5 >$, then we know for sure that the *unk* null of the first tuple can never assume the value 4. Hence any selection (even a weak one) equating the second attribute of r with 4 should discard the first tuple of r. Let $cst(s)$ be a predicate which is satisfied whenever s is a consistent instance. Then we can define our weak selection on a relation r with schema S as follows:

$\tilde{\sigma}_{A=d}(r) = \{t' \mid \exists t \in r : t[A] \in \{d, unk\ , open\ \} \wedge t'[A] == d \wedge t'[D - A] == t[D - A] \wedge cst(r \cup \{t'\})\}.$

$\tilde{\sigma}_{A=B}(r) = \{t' \mid \exists t \in r : (t[A] = t[B]) =/= \text{F} \wedge t'[A] == t'[B] == min(t[A], t[B]) \wedge t'[D - AB] == t[D - AB] \wedge cst(r \cup \{t'\})\}$

where *min* denotes the minimum of two values according to the ordering $\leq$ defined by $\{dne\ , nonnulls\} < unk\ < open$.

Cartesian product $\times$. The cartesian product can be defined in a similar way as the standard one.

Projection π. The projection can be defined in a similar way as the standard one. However, the execution of the π operator may lead to an inconsistent result. Consider, for example the instance r_a of Section 3. If we project on the TEL attribute, then we get an inconsistent relation containing the three tuples $< 5511 >$, $< dne\ >$, and $< unk\ >$. There are different ways to circumvent this problem: One is to require that the projected attributes always contain the primary key of each relation and to forbid that nulls occur in the primary key columns. Another (and more appealing) way is to automatically eliminate all tuples with *dne* nulls which are responsible for inconsistencies from the result of a projection.

A second problem with projection is the elimination of duplicates when projected tuples contain *unk* nulls. If several identical tuples, e.g. of the form $< a, unk >$ are generated during a projection, we propose to eliminate all but one of these tuples and to add the tuple $< a, open >$ to the result.

Union $\cup$ The union operator can be defined in a similar way as the standard one using the *symbolic equality* to eliminate duplicates. As for the projection operator, execution of a union operator may result in inconsistent relation instances.

Join $\bowtie$ Join is a derived operator. It can be defined as a selection applied to a cartesian product of two relation instances. Since we have defined two types of selection, a strong and weak one, we can define a strong and a weak join accordingly.

Difference -,Intersection $\cap$ The definition of these operators is similar to the standard one using the *symbolic equality*. However, problems can arise with redundant relations. For example, if the application of a set difference or of an intersection eliminates the third tuple from instance r_7 of section 5, then the second tuple is no more redundant. Therefore, we must make sure that the operands of these operations are nonredundant. This can be obtained either by eliminating redudant tuples before applying the operations or by imposing some restrictive sufficient conditions on the operands which make sure that they are nonredundant. One such condition is, for instance, that only *dne* and nonnull values are allowed to appear as field values in the operands.

7. Open Problems and Possible Extensions

The most significant contribution of this paper is the presentation and semantical specification of a new approach for binding nulls into the relational model. Several problems, however, have to be resolved until our ideas can be fruitfully applied. The most important of these problems are:

a) The consistency problem when *finite* domains are considered.

b) The problem of finding simple syntactic criteria for nonredundancy of relation instances under both assumptions, finite and infinite domains.

c) The interaction of constraints, such as FDs, MVDs, JDs, etc. with our model of nulls.

d) The adequacy problem of relational operators. We will extend our approach to cover *partitioned* relations with *sure* and *uncertain* tuples [Bisk83, Mai83]. On such relations, the generalisations of the classical relational operators can be defined more adequately.

We also believe that our approach is well suited for being applied in the context of nested relations (see section 3), where nulls have been considered so far only under the open world assumption [RKS85, GZC87].

Acknowledgments

Roberto Zicari has been partially sponsored by the ESPRIT project 432 "METEOR" and has received a travel grant from the Italian Embassy in Vienna, Austria. We are grateful to Peter Pistor, Michael Schrefl, and Marco Ferrario who gave us useful comments and suggestions on the problem of handling null values, as well as to the referees who helped us improving the original version of the paper.

References

[AbBi84] S Abiteboul, N. Bidoit, Non First Normal Form Relation to Represent Hierarchically Organised Data, Proc. 3rd ACM SIGACT-SIGMOD, 1984.

[ANSI75] ANSI/X3/SPARC, Study group on data base management systems: interim report, ACM FDT, 1975.

[BaSp81] F. Bancilhon, N. Spyratos, Update Semantics of Relational Views, ACM TODS, vol.6,no.4,December 1981.

[Bisk81] J.Biskup, A Formal Approach to Null Values in Database Relations, in Advances in Database Theory, Vol. I, (H.Gallaire, J. Minker, J.M. Nicolas) , Plenum Press, New York, 1981.

[Bisk83] J.Biskup, A Foundation of Codd's Relational Maybe Operations, ACM TODS 8:4, 1983, pp.608-636.

[Codd79] E.F. Codd, Extending the database relational model to capture more meaning, ACM TODS 4(4), 1979.

[Codd86] E.F. Codd, Missing Information (Applicable and Inapplicable) in Relational Databases, SIGMOD RECORD, vol.15, no.4, December 1986.

[Codd87] E.F. Codd, More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable information), SIGMOD RECORD, vol.16, no.1, March 1987.

[Dad86] P. Dadam et al., A DBMS to Support Extended NF2 Relations: An Integrated View on Flat Tables and Hierarchies, Proc. ACM Sigmod, 1986, Washington D.C.

[FiVG85] P.Fischer, D. van Gucht, Determining When a Structure is a Nested Relation, Proc. 11th VLDB Conf., Stockholm, Aug. 1985.

[Got82] G. Gottlob, Semantic Representation of Logical Operators of Programming Languages by means of three-valued truth tables, in Proc. IEEE 12th International Symposium on Multiple-valued logic, Paris, May 25-27, 1982.

[GPZ88] G. Gottlob, P. Paolini, R. Zicari, Properties and Update Semantics of Consistent Views, ACM TODS (to appear), also available as Report no. 88-002, 1988, Politecnico di Milano, Milano.

[GZC87] R.H. Gueting, R. Zicari, D.M.Choy, An Algebra for Structured Office Documents, IBM Almaden Research Report RJ 5559(56648), San Jose,CA, 1987 (to appear in ACM TOOIS). (submitted for publication).

[Im84] T. Imielinski ,On Algebraic Query Processing in Logical Databases, in Advances in Database Theory, vol. II, (H. Gallaire, J. MInker, J.M. Nicolas eds.), Plenum Press, New York, 1984.

[ImLp81] T. Imielinski, and W. Lipski, On Representing Incomplete Information in a Relational Database, Proc. 7th VLDB, Cannes, France, 1981.

[KeW85] A.M. Keller, M.Winslett Wilkins, On the Use of an Extended Relational Model to Handle Changing Incomplete Information, IEEE-TSE, vol.7, July 1985.

[Li79] Y.E. Lien, Multivalued Dependencies with Null Values in Relational Databases, Proc. 5th VLDB, Rio de Janeiro, 1979.

[Lp79] W. Lipski, On Semantic Issue Connected with Incomplete Information Databases, ACM TODS 4(3), 1979.

[Mai83] D. Maier, The Theory of Relational Databases, Computer Science Press, Rockville, MD, 1983, Chapter 12.

[Reit78] R. Reiter, On Closed World Databases Logic and Databases, (H. Gallaire, Minker, J.M. Nicolas, eds.), Plenum, New York, 1978.

[Reit86] R. Reiter A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values, JACM, vol.33,no.2, April 1986.

[Res69] N. Rescher, Many Valued Logic , New YOrk, McGraw Hill, 1969.

[RKS85] M.A. Roth, H.F. Korth, A. Silberschatz, Null Values in non1NF Relational Databases, Report TR-85-32, University of Texas at Austin, July 1985.

[ScPi82] H-J. Schek, P. Pistor, Data Structures for an Integrated Data Base Management and Information Retrieval System, Proc. VLDB Conf., Mexico, Sept.1982.

[ScSc86] H-J. Scheck, M.H. Scholl, An Algebra for the Relational Model with Relation Valued Attributes, Information Systems,vol.11, no.2,1986.

[Sno86] R. Snodgrass, Temporal Databases, IEEE Computer, September, 1986.

[Vas79] Y. Vassiliou, Null Values in Database Management Systems: A Denotational Semantics Approach, Proc. ACM SOGMOD, Boston, 1979.

[Wo82] E. Wong, A Statistical Approach to Incomplete Information in Database Systems, ACM TODS 7(3), 1982.

[Zan83] C. Zaniolo, A Formal Treatment of Nonexistent Values in Database Relations, Bell Laboratories, unpublished, 1983

[Zan84] C. Zaniolo, Database Relations with Null Values, Journal of Computer and System Sciences, 28, 1984.